The Prague Bulletin of Mathematical Linguistics
NUMBER 105 APRIL 2016 51–61

CloudLM: a Cloud-based Language Model for Machine Translation

Jorge Ferrández-Tordera[a], Sergio Ortiz-Rojas[a], Antonio Toral[b]

[a] Prompsit Language Engineering S.L., Elche, Spain
[b] ADAPT Centre, School of Computing, Dublin City University, Ireland

Abstract

Language models (LMs) are an essential element in statistical approaches to natural language processing for tasks such as speech recognition and machine translation (MT). The advent of big data leads to the availability of massive amounts of data to build LMs, and in fact, for the most prominent languages, using current techniques and hardware, it is not feasible to train LMs with all the data available nowadays. At the same time, it has been shown that the more data is used for a LM the better the performance, e.g. for MT, without any indication yet of reaching a plateau. This paper presents CloudLM, an open-source cloud-based LM intended for MT, which allows to query distributed LMs. CloudLM relies on Apache Solr and provides the functionality of state-of-the-art language modelling (it builds upon KenLM), while allowing to query massive LMs (as the use of local memory is drastically reduced), at the expense of slower decoding speed.

1. Introduction

Language models (LMs) are an essential element in statistical approaches to natural language processing for tasks such as speech recognition and machine translation (MT). The advent of big data leads to the availability of massive amounts of monolingual data, which could be used to build LMs. In fact, for the most prominent languages, using current techniques and hardware, it is not feasible to train LMs with all the data available nowadays. At the same time, it has been shown that the more data is used for a LM the better the performance, e.g. for MT, without any indication yet of reaching a plateau (Brants et al., 2007).

 Corresponding author: jferrandez@prompsit.com
Cite as: Jorge Ferrández-Tordera, Sergio Ortiz-Rojas, Antonio Toral. CloudLM: a Cloud-based Language Model for Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 105, 2016, pp. 51–61.
doi: 10.1515/pralin-2016-0002.

Our aim in this paper is to build a cloud-based LM architecture, which would allow to query distributed LMs on massive amounts of data. Our architecture is called CloudLM, it is open-source, it is integrated in the Moses MT toolkit[1] and is based on Apache Solr.[2]

The rest of the paper is organised as follows. In Section 2 we provide an overview of the state-of-the-art in huge LMs. Next, Section 3 details our architecture. This is followed by a step-by-step guide to CloudLM in Section 4 and its evaluation in terms of efficiency in Section 5. Finally, we conclude and outline avenues of future work in Section 6.

2. Background

Brants et al. (2007) presented a distributed architecture with the aim of being able to use big data to train LMs. They trained a LM on 2 trillion tokens of text with simplified smoothing, resulting in a 5-gram language model size of 300 billion n-grams. The infrastructure used in their experiment involved 1,500 machines and took 1 day to build the LM. It is worth mentioning that the infrastructure is scalable, so one could use more machines to train LMs on larger amounts of data and/or LMs of higher n-gram orders.

Talbot and Osborne (2007) investigate the use of the Bloom filter, a randomised data structure, to build n-gram-based LMs. Compared to conventional n-gram-based LMs, this approach results in considerably smaller LMs, at the expense, however, of slower decoding. This approach has been implemented in RandLM,[3] which supports distributed LMs.

More recent work explores the training of huge LMs on single machines (Heafield et al., 2013). The authors build a LM on 126 billion tokens, with the training taking 123 GB of RAM, 2.8 days wall time, and 5.4 CPU days. A machine with 1 TB RAM was required to tune an MT system that uses this LM (Durrani et al., 2014).

Memory mapping has been used to reduce the amount of memory needed by huge LMs, at the expense of slower MT decoding speed (Federico and Cettolo, 2007). In the experiments conducted in that work, memory mapping led to decrease the amount of memory required in half at the cost of 44% slower decoding.

The most related previous work to ours is Brants et al. (2007). There are two main differences though: (i) that work relied on a simplified smoothing technique to enhance efficiency while CloudLM uses state-of-the-art smoothing techniques and (ii) our work is open-source and is integrated in the Moses statistical MT toolkit.

[1] https://github.com/jferrandez/mosesdecoder/tree/cache-cloudlm

[2] http://lucene.apache.org/solr/

[3] http://randlm.sourceforge.net/

3. Architecture

This section describes the architecture of CloudLM. First we cover the representation of LMs in Apache Solr (Section 3.1). Then we detail the implementation of CloudLM in the Moses toolkit (Section 3.2). Finally, we describe two efficiency enhancements that have been added to CloudLM, a cache and queries (Section 3.3).

3.1. LMs in Solr

In order to have LMs in Solr, we need to represent in a Solr schema the fields of an ARPA LM entry,[4] namely: (i) the n-gram, (ii) its probability, (iii) its back-off weight and (iv) its order. We define these fields in a Solr schema as shown in Figure 1.

```
<field name="ngram"   type="string" indexed="true"  stored="true"/>
<field name="prob"    type="float"  indexed="false" stored="true"/>
<field name="backoff" type="float"  indexed="false" stored="true"/>
<field name="order"   type="int"    indexed="true"  stored="true"/>
```

Figure 1. ARPA fields in Solr schema

The fields `ngram` and `order` are indexed (`indexed="true"`) as those are the ones we use to query the LM. All the fields are stored (`stored="true"`) meaning that they can be returned by queries.

3.2. Cloud-based LM in Moses

CloudLM is implemented as a new module in Moses that builds upon KenLM (Heafield, 2011). In short, we adapt KenLM's functions that query n-grams on ARPA or binary files so that they query our cloud-based model instead and we remove any other files that are not required for querying LMs (e.g. build and binarise LMs, trie models, quantise, etc.). As a result, given a query and a LM, the output produced by CloudLM and KenLM are exactly the same.

CloudLM provides two main advantages as a result of its distributed nature: (i) there is no loading time and (ii) memory requirements in the machine where decoding takes place are considerably reduced. That said, there is an important disadvantage, in that its use results in slower MT decoding speed.

[4] `http://www1.icsi.berkeley.edu/Speech/docs/HTKBook3.2/node213_mn.html`

3.3. Efficiency Enhancements

In order to mitigate the main disadvantage of CloudLM (its lower querying speed), we implement two efficiency enhancements, a cache (Section 3.3.1) and a block query (Section 3.3.2).

3.3.1. Cache

Caches are known to be useful in any network dependent process (thus subject to high latency) that requests repeated queries. In CloudLM we implement a cache in order to avoid several queries requesting the probability for the same n-gram.

Intuitively, the advantage of the cache is that it should save time due to network latency. However, the data stored in the cache structure should lead to higher requirements of memory.

Our selected cache strategy is least recently used (LRU), in which the least recently used items are discarded first. In the way that Moses queries the LM, LRU guarantees that the most recently requested n-grams will be found in the cache.

3.3.2. Block Query

As we adapt KenLM querying functions only with respect to the repository where the LM is stored (from local files to Solr), queries are still submitted individually for each n-gram. For example, given the 2-gram "we are", three queries would be submitted to the LM: one for the bi-gram "we are" and two for the 1-grams "we" and "are". Our first approach for using the cache is to store the probability returned for this 2-gram.

In order to minimise the amount of queries sent to Solr (and saving network latency), we implement a block n-grams query. When the LM receives a phrase, we extract all its possible n-grams and prepare a query that contains them all. For instance, for the previous example, we prepare a query with the 2-gram "we are" and the 1-grams "we" and "are". In this way we can retrieve the three probabilities with one single query.

4. Step-by-Step

In this section we provide a step-by-step guide to use CloudLM in Moses. We assume we have a Moses system (Koehn et al., 2007) trained (translation and reordering models), e.g. according to Moses baseline guide,[5] an LM ready in ARPA format, e.g. trained with KenLM, and an installation of Apache Solr. The steps are as follows:

1. Configure Solr.

[5] http://www.statmt.org/moses/?n=Moses.Baseline

The LM can be placed in the local machine, in a remote one, or be distributed across a number of machines. We cover each of these cases in the following:

- Local machine. While the main advantage of using Solr relies in its distributed nature, we can still use it locally, where Solr's advantage will be its lower use of memory (as the LM is not loaded completely in RAM).
- Remote machine. In this case Solr is used from one remote machine. This can be useful when the local machine does not have enough resources for the LM but we have access to a remote machine with enough resources.
- Distributed architecture. Solr allows to have the LM distributed across a number of machines.[6] This can be useful when we have access to a number of remote machines and we have to deal with a huge LM that does not fit on any of those machines alone.

2. Upload LM to Solr. This is done with a script included with CloudLM that reads an ARPA file, converts it to Solr Schema (cf. Section 3.1) and uploads it to a Solr installation.

```
python add-language-model-from-arpa.py \
       http://localhost:8983/solr lm.arpa
```

3. Include CloudLM in Moses' configuration (ini file). The format is very similar to that of KenLM, the only three differences being that (i) the LM type is `CLOUDLM` (instead of `KENLM`), that (ii) the LM is indicated by means of a URL (instead of a local path) and that (iii) there is a binary variable to indicate whether or not to use a cache (cf. Section 3.3.1).

```
CLOUDLM name=LM0 factor=0 order=4 \
        num-features=1 cache=0 url=localhost:8983
```

From this point onward, we can proceed with tuning and decoding with Moses as one would normally do.

5. Experiments

In this section we conduct a set of experiments in order to measure efficiency of CloudLM in terms of computational resources (real time and memory) in the task of statistical MT. First, we detail the experimental setting (Section 5.1) and then we present the results for three experiments (Section 5.2) where we measure (i) the effect of the efficiency enhancements on top of CloudLM, (ii) the effect of network latency and finally we (iii) compare the efficiency of CloudLM to that of a state-of-the-art *local* LM, KenLM.

[6] `https://cwiki.apache.org/confluence/display/solr/SolrCloud`

5.1. Experimental Setting

The MT systems built for our experiments fall into the statistical phrase-based paradigm and they are built with Moses version 3 following the baseline system guideline.[7] All the systems are trained on the Europarl v7 (Koehn, 2005) parallel corpus for the language direction English-to-Spanish. All the LMs are built on the Spanish side of that parallel corpus. [8] We use these MT systems to decode subsets (1, 10 and 100 sentences) of the test set from WMT13.[9]

We use both a local and a remote machine in our experiments.[10] The local machine has a 8-core i7-3632QM CPU at 2.20GHz, 16GB RAM and a SATA 3.0 500GB hard drive. The remote machine has 4-core Q8200 CPU at 2.33GHz, 4GB RAM and a SATA 3.0 1TB hard drive.

5.2. Results

In all the experiments below we measure the peak of memory used and real time required to translate the first 1, 10 and 100 sentences of the testset with the different systems evaluated.

5.2.1. Effect of Enhancement Additions

In this experiment we measure the effect of the efficiency enhancements that have been added to CloudLM, namely the cache (cf. Section 3.3.1) and block queries (cf. Section 3.3.2). We build three systems where the LMs are built with CloudLM using different settings: stock (reported in results below as S), with cache (WC) and with both cache and block queries (WCBQ). All the LMs are stored locally.

Figure 2 reports the real time and Moses' memory peak required by each system to decode the first 1, 10 and 100 sentences from the test set. The use of cache, as expected, results in a notable reduction in time but also increases memory usage. For 1 sentence, using cache reduces the time by around 70% and memory used augments by 20%. These figures increase with the number of sentences decoded; with 100 sentences the use of cache reduces the time required by 89% while memory used increments by 195%. On its turn, the use of block queries (WCBQ) provides a slight time advantage when decoding 1 sentence (9% faster), but it is slower for 10 and 100 sentences. We are currently investigating the causes for this.

Table 1 provides further details regarding the use of cache and block queries. It shows the total number of requests submitted to Solr (column # requests), the number

[7] `http://www.statmt.org/moses/?n=Moses.Baseline`

[8] The existing model indexed in Solr takes 1.95 GB. The original binarized ARPA file amounts to 830 MB.

[9] `http://www.statmt.org/wmt13/translation-task.html`

[10] Before each run the machines were rebooted to ensure data from the previous run is not leveraged from the disk cache.

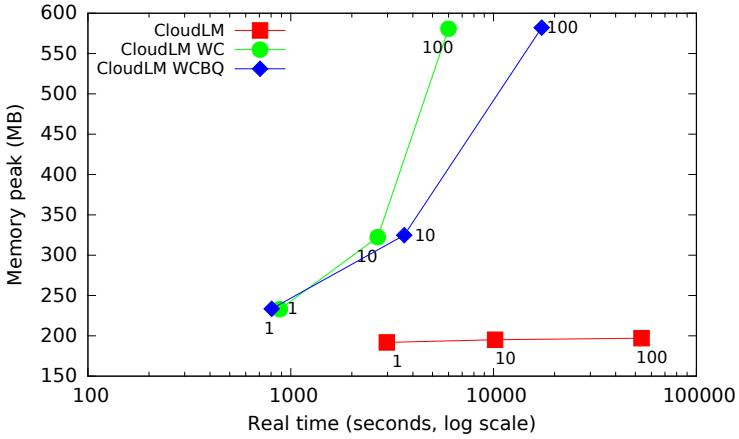

Figure 2. Effect of enhancement additions

of queries that are stored in the cache (# insertions), the number of lookups in the cache (# lookups) and the percentage of successful lookups (% found), i.e. the cache contains the query requested and hence the query is not submitted to Solr. The use of the cache reduces drastically the number of queries sent to Solr, even when translating just one sentence this number is reduced by 85%.The use of block queries reduces even more the amount of queries sent, as the percentage of queries found in the cache is even higher (e.g. 99.8% for 1 sentence).

# sents.	System	# requests	# inserts	# lookups	% found
1	S	1,779,225	0	0	0
1	WC	264,851	264,851	1,779,225	85.11
1	WCBQ	206,160	264,851	1,779,225	99.80
10	S	7,067,343	0	0	0
10	WC	822,627	822,627	7,067,343	88.36
10	WCBQ	929,020	822,627	7,067,343	98.21
100	S	22,417,996	0	0	0
100	WC	2,417,593	2,417,593	22,417,996	89.21
100	WCBQ	4,493,867	2,417,593	22,417,996	94.45

Table 1. Effects of the use of cache and block queries with CloudLM.

5.2.2. Effect of Network Latency

In this experiment we measure the effect of network latency. Clearly, an advantage of CloudLM relies in the fact that it allows us to use LMs placed in remote machines. Accessing them, though, penalises efficiency as each query is subject to network latency.

We use two systems, both using CloudLM with cache. One of the systems accesses the LM locally while the other accesses it from a remote machine in the local network.

Figure 3 reports the real time and memory peak required by each system to decode different amounts of sentences. Network latency affects efficiency quite drastically; accessing the LM from a remote machine results in decoding speed an order of magnitude slower. We measured the average latency in the local and remote machines used in this experiment. The figures were 0.04 milliseconds for the local machine and 0.277 for the remote one.

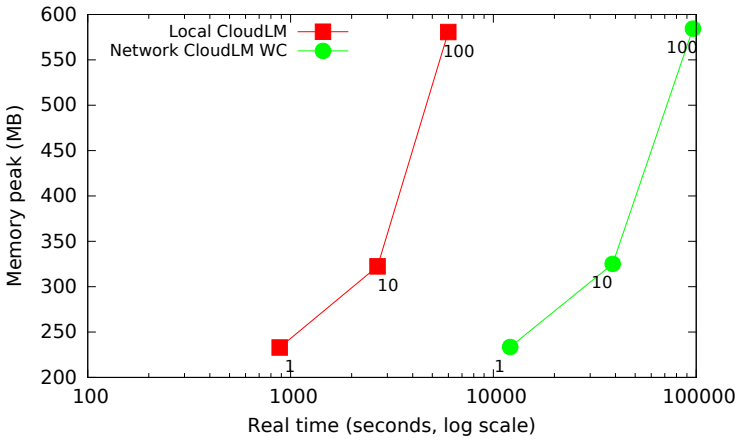

Figure 3. Effect of network latency

5.2.3. Comparison to a *local* LM

Finally, we compare, in terms of efficiency, CloudLM to a state-of-the-art *local* LM, KenLM. We have three systems, one with CloudLM (with cache), and two with KenLM (with and without loading on demand, reported in the results as lazy KenLM and KenLM respectively). All LMs are stored locally.

Figure 4 shows the results. CloudLM reduces notably the amount of memory required at the expense of the decoding speed becoming between one and two orders

of magnitude higher. For one sentence, CloudLM is 70 times slower (240 compared to KenLM on demand) and reduces the amount of memory required by 77% (65% compared to on demand). As we add more sentences the differences on both speed and memory shrink, with CloudLM being 33 times slower (68 compared to the on demand version of KenLM) and reducing the amount of memory by 46% (45% compared to KenLM on demand) for 100 sentences.

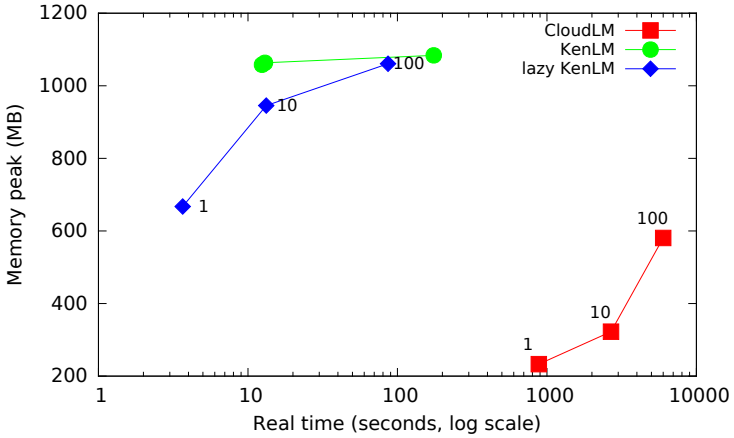

Figure 4. Comparison of CloudLM to a local LM

6. Conclusions and Future Work

This paper has presented CloudLM, an open-source cloud-based LM that allows to build distributed LMs for their use in e.g. MT. CloudLM is based on Apache Solr and KenLM, providing the functionality of the latter in a distributed environment. The focus of our work so far has been on providing a stable and robust implementation that can be extended upon to make it more efficient.

The current implementation uses a simple cache model (LRU) and can send joint queries in order to diminish the efficiency penalty posed by network latency. We have evaluated CloudLM in terms of efficiency to measure the effect of the efficiency additions, the effect of latency and finally to compare its use of resources compared to a state-of-the-art *local* LM.

We envisage two main lines of future work. First, development work to enhance efficiency. We have several ideas in this regard, such as keeping the connection alive between Moses and Solr (so that a new query does not need to re-open the connection) and using more advance cache strategies. The efficiency bottleneck in a synchronous

distributed architecture like ours has to do with the network latency. Hence, we propose to have an asynchronous connection instead, so that Moses does not need to wait for each response from Solr. This, however, is far from straightforward as it would entail deeper modifications to the MT decoder.

Our second line of future work has to do with the evaluation of CloudLM for huge LMs. The evaluation in the current paper can be considered as proof-of-concept, as we have dealt with a rather small LM (around 2 million sentence pairs).

Finally, we would like to compare CloudLM to other approaches that use distributed LMs in Moses (Federmann, 2007; Talbot and Osborne, 2007). Such an evaluation would not be purely efficiency-based (e.g. decoding time, memory used) but also would take into account the final translation quality achieved as some of these approaches use different modelling techniques (e.g. Bloom filter in RandLM).

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement PIAP-GA-2012-324414 (Abu-MaTran).

Bibliography

Brants, Thorsten, Ashok C. Popat, Peng Xu, Franz Josef Och, and Jeffrey Dean. Large Language Models in Machine Translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, 2007. URL `http://www.aclweb.org/anthology/D/D07/D07-1090`.

Durrani, Nadir, Barry Haddow, Philipp Koehn, and Kenneth Heafield. Edinburgh's Phrase-based Machine Translation Systems for WMT-14. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 97–104, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W14/W14-3309`.

Federico, Marcello and Mauro Cettolo. Efficient Handling of N-gram Language Models for Statistical Machine Translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 88–95, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W07/W07-0212`.

Federmann, Christian. Very Large Language Models for Machine Translation. Master's thesis, Saarland University, Saarbrücken, Germany, July 2007. URL `http://www.cfedermann.de/pdf/diploma-thesis.pdf`.

Heafield, Kenneth. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom, July 2011. URL `http://kheafield.com/professional/avenue/kenlm.pdf`.

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the*

Association for Computational Linguistics, pages 690–696, Sofia, Bulgaria, August 2013. URL http://kheafield.com/professional/edinburgh/estimate_paper.pdf.

Koehn, Philipp. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86. Citeseer, 2005.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1557769.1557821.

Talbot, David and Miles Osborne. Smoothed Bloom Filter Language Models: Tera-Scale LMs on the Cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 468–476, 2007. URL http://www.aclweb.org/anthology/D/D07/D07-1049.

Address for correspondence:
Jorge Ferrández-Tordera
jferrandez@prompsit.com
Avenida Universidad s/n Edificio Quorum III - Prompsit,
Elche, Alicante, Spain. 03202